Notes to Self

- I don't think we'll have access to any of the Solver pieces (for example the search tree), since we'll want this to work with built-in solvers too.
- PriorityFlawHandler?: PriorityMinPerturb? and MinPerturbMerge?
- Have their own threat manager

TODO

- See below question about Solver::reset
- Talk to Mike about how he might incorporate what DynamicEUROPA did into current architecture
- Get good example that exercises various capabilities of system so we can test anything we implement
- Overview of existing research (ask Paul and David if they know of any, for starters). Javier is doing this
- Move the existing DynamicEuropa implementation over (just move the code over - should be easy, ONCE I understand the code!)
- Try to implement a more generic or better version for the general case

DynamicEUROPA Approach

Described here to best of my understanding. See also their MAPGEN paper from ICAPS 05 (attached).

Their approach is three-pronged:

1. **NDDL Changes**
2. **Code wrapped around core EUROPA**
3. **Solver changes:** They use the built-in solver but:
    1. *Threats* handled with custom PriorityFlawHandler (registered as PriorityMinPerturb) which uses MinPerturb? decision point to resolve threats. This decision point uses reference times to determine order of merges to suggest (for each choice, it looks at how far the token would have to move from the reference time, and prefers small moves).
    2. *Open conditions* handled
    3. *Unbound (temporal) variables* are left unbound (ie filtered by SolverConfig?.xml), so that the above code (fixViolations etc) can bound them according to the reference schedule.

Other things they've done are intertwined with the above:

- Augment nddl predicates with 'scheduled' boolean and wrap all predicate constraints and subgoals within 'if(scheduled==true)' guard to be able to solve over-subscribed problems (ie activities can be left unscheduled to get a feasible plan)

Questions

- How does Dynamic EUROPA implementation work:
    - ♦ Do they only consider temporal variables?
    - ♦ What about removing/adding tokens to the plan? Do they ever do it?
- Which do we want:
    - ♦ Ability to continue where we left off (ie everything still in the database etc)?
    - ♦ Start from scratch (ie solve a new problem given some stored info about our previous solution)?

- Does plan database record time stamps for data (ie can you recreate search by looking at it)?
- Does existing chronological search handle inconsistencies (or just the underlying structures)?

Example Perturbations

- A variable range is further restricted (temporal variables are one example).
- A constraint is updated or added
- Goal is changed
- Initial state changes
- Resource information is updated
- Indirect results of perturbation could be:
    - Don't need a token anymore
    - Merge not possible anymore
    - New merge is possible
    - Optional goal now reachable (or now no longer reachable)

Brainstorming Notes

- Note that a min perturb solver subsumes a solver that resolves conflicts - perhaps focus on the latter first (should be easier, but isn't obvious how to do even that).
- QUESTION:
- Undoing decisions will result in new activity ids next time, so we can use them to store reference data (and if variable contains preference, it would be lost!)
- As DynamicEUROPA guys do, it is possible to restart the solver, without restarting the current state (reset eliminates decision stackv! TODO: Is this true, or does eliminating stack undo those things??? SOS

- Why not simply take advantage of the ability to be in an inconsistent state and continue whatever search we're doing? Either start in the previous state, or just load the previous state in, if necessary.
- A modification of existing search where:
    - Each step does something that matches the final state in the old situation. If it can't, do nothing and branch on a different variable (ie will make everything how it was as much as possible, before starting to set things as close as possible).
    - Variable issues (what value to choose) are different than object issues (whether to activate, merge, etc)
- Save a backup copy of current plan database. Then start from scratch. Implement something like existing chronological search:
    - At decision point, make choices to mimic what's in saved database
    - If that makes things inconsistent, skip handling that flaw for now (ie recreate as much as possible that is identical)
    - Once we get to a point where we must deviate in some way from previous plan, just start the usual search (or any other algorithm) from there